Class 1E Software Verification and Validation: Past, Present, and Future[1]

Warren L. Persons and J. Dennis Lawrence
Lawrence Livermore National Laboratory
Fission Energy and Systems Safety Program
Computer Safety & Reliability Group

This paper[2] discusses work in progress that addresses software verification and validation (V&V) as it takes place during the full software life cycle of safety-critical software. The paper begins with a brief overview of the task description and discussion of the historical evolution of software V&V. A new perspective is presented which shows the entire verification and validation process from the viewpoints of a software developer, product assurance engineer, independent V&V auditor, and government regulator. An account of the experience of the field test of the Verification Audit Plan and Report generated from the V&V Guidelines is presented along with sample checklists and lessons learned from the verification audit experience. Then, an approach to automating the V&V Guidelines is introduced. The paper concludes with a glossary and bibliography.

1. INTRODUCTION

1.1. Task Description

The work in progress described in this paper builds upon previous work performed by the United States Nuclear Regulatory Commission (NRC), Office of Nuclear Regulatory Research (RES) and the Office of Nuclear Reactor Regulation (NRR) in the areas of verification and validation (V&V) guidelines for the evaluation of safety-critical software. The purpose of this effort is to field test the audit process and principles put forth in these guidelines. The major thrusts of this effort involve reviewing the V&V Guidelines, applying the guidelines to a verification and validation audit of Class 1E software, performing a cost/benefit analysis of computerizing the audit guidelines on a laptop computer to serve as an aid for reviewers, meeting with the NRC and guideline developers to discuss proposed modifications, and reporting on the work performed.

[1] This work was supported by the United States Nuclear Regulatory Commission under a Memorandum of Understanding with the U.S. Department of Energy, and performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

[2] The authors appreciate the guidance provided by the many discussions with Mr. Leo Beltracchi; Nuclear Regulatory Commission, Office of Nuclear Regulatory Research, that led to concepts presented in this paper.

1.2. What is Class 1E Software?

IEEE Standard 379, *Application of the Single Failure Criterion to Nuclear Power Generating Station Class 1E Systems* (1977), defines Class 1E as, “The safety classification of the electric equipment and systems that are essential to emergency reactor shutdown, containment isolation, reactor core cooling, and containment and reactor heat removal, or are otherwise essential in preventing significant release of radioactive material to the environment.” In this paper, software used in Class 1E systems is referred to as Class 1E software. Class 1E software is currently being used in nuclear power plants and applications including (but not limited to) reactor trip systems and emergency generator load sequencers.

1.3. What is Software Verification and Validation?

The terms “verification,” “validation,” and “verification and validation” abound in the software literature and are used with various explicit or implicit meanings. Some of the diversification may be rooted in a particular author’s need to customize the terminology so that it is appropriate for a specialized application area. Another reason may be the type or criticality of the software application; e.g., software applications range from a simple spreadsheet used to track hours an individual works on a project, to highly reliable flight control software used for the space shuttle. Intuitively, there is a difference in what V&V should mean in each case and in the amount of effort that should be applied to V&V activities for these extreme application types.

There is some consensus among software engineers that the activities of verification and validation are focused on the determination of whether the software performs its intended function and has the required quality attributes. This body believes that V&V as a formal discipline is near the midpoint of its development. As such, specific formal boundaries have yet to be established to define what the extent of the V&V functional activities should be. However, a large body of opinion agrees that V&V is one of the techniques used to help identify, assess, and manage risks in software development projects and can be carried out at varying levels of rigor, depending on the nature of the application and the risks involved in the development activity. The utmost rigor is required when one of the risks is safety.

The terms verification, validation, or verification and validation are used in this paper in accordance with the following IEEE Standard 610.12-1990 definitions:

Verification. The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of the phase.

Validation. The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.

Verification and validation. The process of determining whether the requirements for a system or component are complete and correct, the products of each development phase fulfill the requirements or conditions imposed by the previous phase, and the final system or component complies with specified requirements.

1.4. What is the Problem?

Nuclear reactors, like any complex industrial plants, routinely experience equipment or operational failures, some of which could lead to serious consequences. Unlike many industrial plants, one potential consequence of reactor accidents is the release of radiation or radioactive material into the environment. Until the mid 1970s, software was not used in Class 1E systems, but a rapid transition into an era of computer and software control of these Class 1E systems is now occurring.

Many risks exist in the complex process of producing high-quality Class 1E software. Most software projects should be concerned with the risks of failing to meet cost and schedule goals. In some cases, there are additional technological risks due to the use of new hardware, new programming languages, new design techniques, or new software tools. When safety is at issue, special attention must be paid to all of the above risks as well as to the special risks to human life and health, property, and the environment. Software developers must build safe and reliable software products that satisfy the requirements allocated to the software portion of the Class 1E system. In particular, unless the set of software development activities is carefully managed, the complexities and uncertainties inherent in these procedures can cause unnecessary risk, delays, and expenses.

An aspect of the problem is that building any system or any software system is effectively a problem-solving exercise. As such, all of the difficulties associated with problem solving are encountered during the software development activity. In particular, there are limitations in the software engineer’s communication skill, ability, experience, problem understanding, flexibility to view the problem from multiple perspectives — that is, the ability to shift paradigms — as well as normal technical difficulties associated with any problem-solving endeavor. Other components of the problem are specific to software development. First, software development is a labor-intensive, intellectual activity. A noted software expert, Edgar Dijkstra, stated in 1969 that software development is one of the most intellectually challenging activities in which humans can engage. Second, software does not wear out and hence, does not fail in the same manner that hardware does. Software can contain manufacturing or design defects which can produce hazards and failures in operational use. It is quite common for software errors to be very subtle and to be completely overlooked during implementation, but their presence can cause catastrophic failure after years of apparently successful operation.

1.5. What Makes Class 1E Software Different?

Considering the extensive use of computers and software during the last 20 years in various applications, why is there concern over the application of this technology to Class 1E systems? Simply stated, the following Class 1E software attributes raise the level of concern and make it different:

- Potential risk to life, property and environment,
- Safety requirements, and
- High reliability requirements.

There are two central concepts that are key to the development of Class 1E software. First, software safety and reliability considerations cannot be fully understood in isolation from computer hardware and application considerations. Second, the process of engineering reliability and safety into a computer system requires activities to be carried out throughout the software life cycle. Thus, the development, use, and regulation of computer systems in nuclear reactors is a complex issue.

1.6. General Approaches to Increase Confidence in Class 1E Software

There are two general approaches for increasing the confidence of Class 1E software systems, as shown in Figure 1: (1) reducing, if not eliminating, the number of errors introduced during the software development process, and (2) increasing the percentage of overall errors found prior to system installation. Note that Figure 1 shows only one sample translation that occurs during the software development process. In this case, a translation is shown from the requirements, allocated by the Class 1E system to the Class 1E software, to the software design. Similarly, there are translations from design to code, from code to integrated components, and so forth throughout the software development process. Each translation provides opportunities to insert or inject errors and to detect inserted errors. The goal is to insert fewer errors and simultaneously detect more errors as each translation occurs during the software development process. The goal is “error free software;” the focus is on continual process improvement and eliminating errors before they are propagated. For Class 1E software, both methods are appropriate.

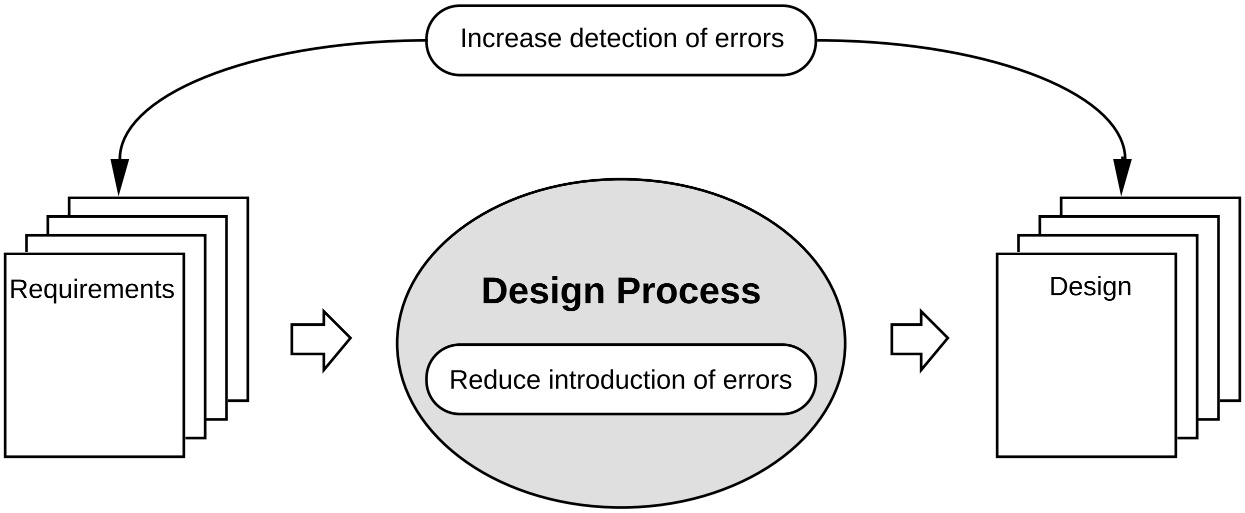


Figure 1. Reduction of Errors Versus Detection of Errors

1.7. Issues Associated with Class 1E Software

If a comparison is made between current electro-mechanical Class 1E systems and software-based Class 1E systems, several issues related to the use of software in these systems can be identified:

- Lack of experience in developing Class 1E software.
- Inability to measure required, ultra-high software reliability.
- Lack of a mathematical basis for safety-critical software construction.
- Difficulty in formally proving software correctness.
- New potential for common-mode failures.
- Lack of operational data.
- Small errors may have significant consequences.

In spite of this, software-based systems may be the only reasonable alternative for replacing aging nuclear reactor protection system components. Traditional analog/relay equipment is becoming much more expensive and, in some cases, is totally unavailable. If substitute equipment is available, it often contains embedded digital hardware and software which share the issues just listed. A large body of opinion, however, believes that with proper use of modern software engineering practices, the number of residual defects in delivered Class 1E software can be reduced to an acceptable minimum, and these remaining defects will not have severe consequences.

Many risks are involved in the development of Class 1E software for use in nuclear power generating stations. For example:

- Delaying an audit or evaluation until the end of the development effort can be very expensive. Extensive industry experience shows that errors are more easily fixed and less expensive to fix if they are found in early development phases.
- Delaying evaluations until after the development effort is complete may require more extensive proof that the safety requirements have been met, and it may actually be impossible at this point to assess the safety of the system.
- Inconsistencies among auditors, or by a single auditor over a period of time, can lead to unsettling differences in evaluation results and required corrective actions.
- Evaluations are inherently labor-intensive procedures. Lack of computer assistance increases the time and effort involved in performing these evaluations, which results in increased costs to both the developer and the regulatory agency.
- A possible result of these uncertainties and increased costs can be less-reliable software, which in turn can increase the difficulty of assessing the safety of the reactor.

2. THE EVOLUTION OF V&V

In the early history of computing, software was produced without a clear written document describing, beforehand, what the software was supposed to do. In some cases the results of a particular software task surprised even those who were responsible for the detailed implementation. Many “features” were discovered as testing and use proceeded. A lack of discipline in the software development process wasted resources and caused considerable customer dissatisfaction. It was discovered by some software vendors that the production of a requirements specification prior to the writing of code led to a better product and reduced costs. But surprises still occurred. Testing was made more formal and it became more meaningful because there was a requirements specification against which to test. The requirements specification also improved as ambiguous and untestable elements were removed from the specifications. Still, the results tended to contain “features” which were undesirable and defects which were discovered only during use.

Discipline in software development became more formalized and the notion of “design” was introduced as a formal step in the software development life cycle. It was recognized by some that software development is an “engineering” discipline and the term “software engineering” was introduced. Quality control and quality assurance issues were addressed as reviews and inspections slowly made their way into the software development life cycle. Configuration management was introduced as software engineering matured.

The notion of a software life cycle became commonplace with activities such as: planning, requirements, design, implementation, test, installation, operation and maintenance, and retirement. The terms “verification” and “validation” (termed collectively V&V) were introduced as part of the engineering discipline. Originally, V&V applied only to the technical products of the software development process. The managerial and product assurance aspects of software development were not originally subjected to V&V.

Both product assurance and configuration management can trace their historical roots to the hardware side of the engineering discipline. On the other hand, V&V came into existence to cope with software and its development. Gradually it was recognized that preventive testing was a part of management’s tool kit for risk management and that testing should have a life cycle of its own. Testing should be planned, analyzed, designed, implemented, executed, and the results recorded. Moreover, each of these activities should be subject to inspections and reviews, with the results of each activity placed under configuration control.

Software projects and products have evolved from small systems that were typically developed by a few people to much larger, more complex systems involving up to several hundred people on the software development team. This change in the characteristics of software projects and products has caused a radical change in the notion of verification and validation. Originally, V&V was very informal and individualized and was focused on testing. Early V&V merely involved the programmer exercising the code that was produced. As the software systems were required to perform more and more functions, and the resulting systems became larger and more complex, the neglect of planning, design and execution of test procedures and test cases led to poor quality and to defective, unreliable software products.

Some companies have discovered that, when risks are high, V&V should apply to all aspects of software development. Companies which produce software for safety-critical applications are beginning to use a much more formal V&V process. This notion of V&V extends beyond the traditional restriction to technical aspects of software development in order to include management and product assurance. This can only be done by an organization which is independent of the developer. Independent V&V (IV&V) is performed on all software products which are part of the software engineering and product assurance activities. In particular, this independent analysis must give considerable attention to identifying, assessing, and managing risks and hazards.

The overall goal of IV&V is to ensure that software quality is achieved. Part of this goal is achieved by providing specific visibility into the entire development process so that management decisions can be made to assure appropriate software quality. IV&V continues to evolve as software development projects change and has become a very powerful management tool.

3. A PERSPECTIVE ON V&V

3.1. Software Evaluation Perspectives

Any software evaluation process can be discussed from several different viewpoints. The viewpoint that is presumed has a considerable effect on the topics discussed, and particularly the emphasis placed on different aspects of the evaluation process. In this discussion, three viewpoints are considered: the regulatory view, the independent verification and validation view, and the software development view. Each viewpoint, as shown in Figure 2, has its own goals, evaluation perspective and activities, which are described below.

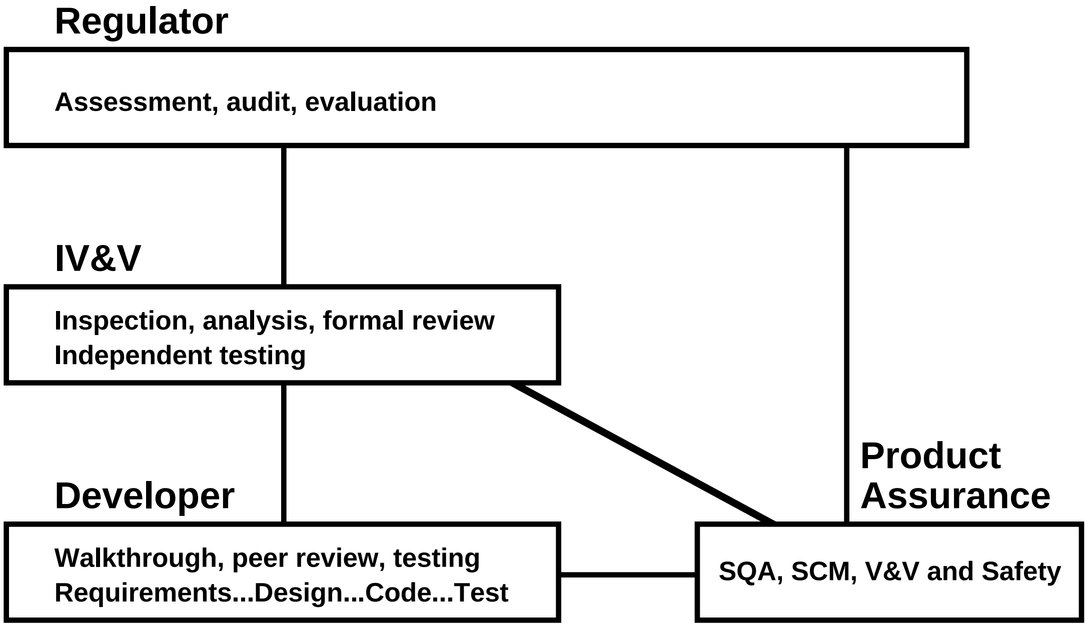

Regulator
Assessment, audit, evaluation
IV&V
Inspection, analysis, formal review
Independent testing
Developer
Walkthrough, peer review, testing
Requirements...Design...Code...Test
Product
Assurance
SQA, SCM, V&V and Safety

Figure 2. Model for Software Evaluation

3.2. Developer’s Viewpoint

In this paper it is assumed that the developer of Class 1E software defines the methods to be used to deliver that software in a project plan, which is subject to regulatory approval. Once approved, the software developer is held accountable for development in accordance with the project plan. This two-step sequence of project plan approval followed by project plan accountability offers both flexibility on the part of the developer and insight on the part of the regulatory agency. The separation of the development function into the equivalent of requirements, design, implementation, integration, validation, installation, and operations and maintenance activities is a fairly standard practice.

The software developer will carry out certain product assurance activities, as suggested in Figure 2. In this paper, the term “product assurance” is used to cover the developer’s activities which relate to V&V, testing, software quality assurance (SQA), software configuration management (SCM), and safety analysis. The developer’s product assurance provides the first objective evidence of the quality of the development effort.

3.3. IV&V Viewpoint

The ultimate result of the software development process, as considered here, is a suite of computer programs and its related documentation. The documentation guides the user, developer, installer, and maintainer of the software throughout the life cycle. These programs must have characteristics such as safety, reliability, performance, usability and function. The purpose of IV&V is to provide independent assurance that the required characteristics have been met by the developer. The importance of IV&V is emphasized in various standards listed in the bibliography at the end of this paper.

3.4. Regulator’s Viewpoint

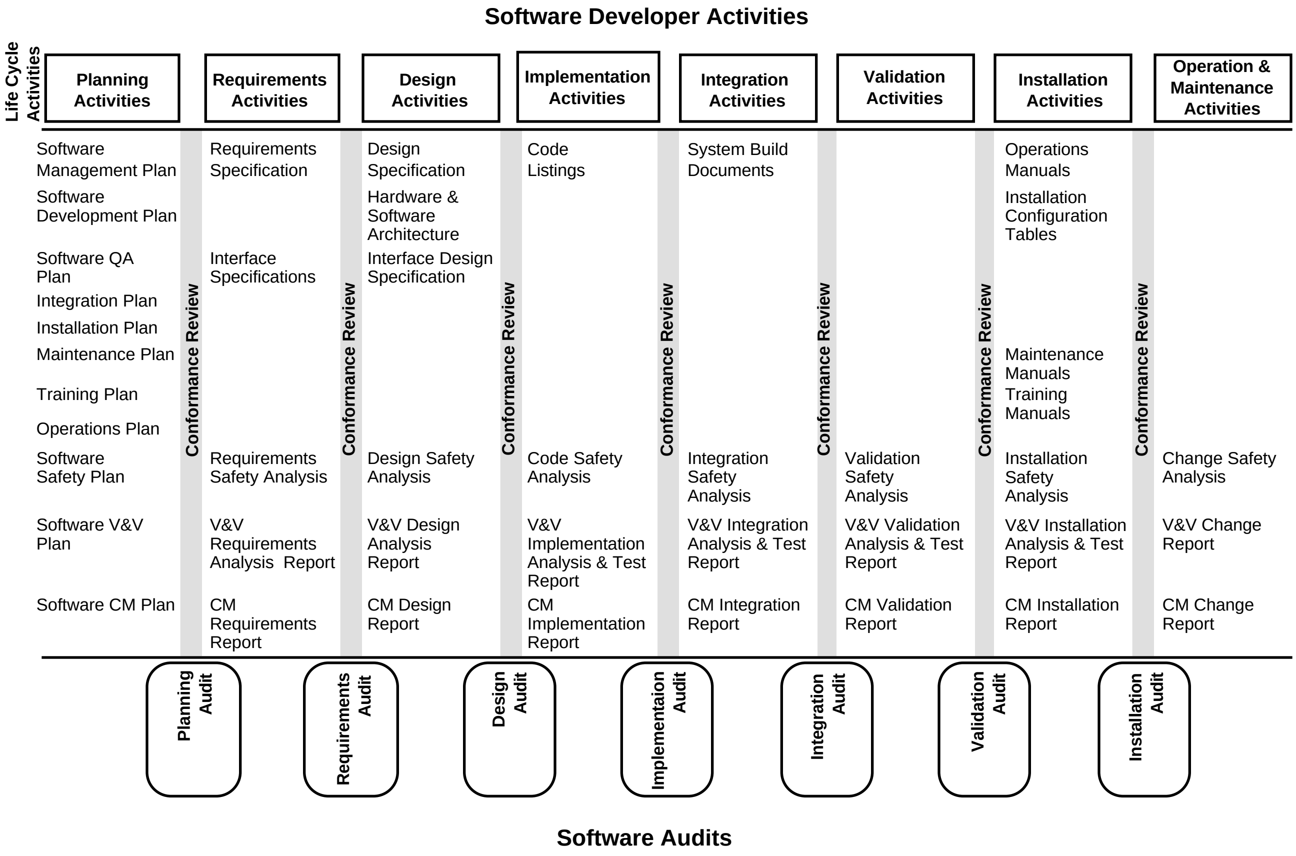
Regulators are required by legislation to license nuclear power generation stations. One aspect of this is to analyze the evaluations performed by the software developer (in the form of product assurance activities) and the IV&V team activities. This may best be done using standards (such as those found in the bibliography) to help evaluators perform assessments or audits in an efficient cost-effective fashion. These audits or assessments should be consistent over time, across companies, and across projects within companies.

4. THE VERIFICATION AUDIT FIELD TEST

4.1. Verification Audit Context

The first major activity on the field test was to review and apply the V&V Guidelines to the development of an audit plan to be used to audit existing Class 1E software components. The V&V Guidelines were provided as NRC-furnished material. This audit was performed at a vendor’s facility in January 1993 and was conducted by an interdisciplinary audit team consisting of personnel from NRC/NRR, SoHaR Incorporated, and Lawrence Livermore National Laboratory. The audit evaluated both the software development process used to develop Class 1E software products that are components of Class 1E systems, and the products of that process.

Figure 3 identifies checkpoints at which software audits of the software development activities, processes, or products can be performed. The number of audits depends, among other things, on the specific software life cycle used by the vendor or software developer. Each audit analyzes the work done relative to that checkpoint. Many reliability, performance, and safety problems can be resolved only by careful design of the software product, so they should be addressed early in the software development process, no matter which life cycle is used. Any errors or oversights can require difficult and expensive retrofits, so they too are best found as early as possible. Consequently, an incremental V&V audit process is believed to be more effective than a single audit or evaluation at the end of the development process. Using multiple audits, problems can be detected early in the software life cycle and corrected before large amounts of resources have been consumed.



Planning Activities	Requirements Activities	Design Activities	Implementation Activities	Integration Activities	Validation Activities	Installation Activities	Operation & Maintenance Activities
Software Management Plan	Requirements Specification	Design Specification	Code Listings	System Build Documents		Operations Manuals	
Software Development Plan		Hardware & Software Architecture				Installation Configuration Tables	
Software QA Plan	Interface Specifications	Interface Design Specification					
Integration Plan							
Installation Plan							
Maintenance Plan						Maintenance Manuals	
Training Plan						Training Manuals	
Operations Plan							
Software Safety Plan	Requirements Safety Analysis	Design Safety Analysis	Code Safety Analysis	Integration Safety Analysis	Validation Safety Analysis	Installation Safety Analysis	Change Safety Analysis
Software V&V Plan	V&V Requirements Analysis Report	V&V Design Analysis Report	V&V Implementation Analysis & Test Report	V&V Integration Analysis & Test Report	V&V Validation Analysis & Test Report	V&V Installation Analysis & Test Report	V&V Change Report
Software CM Plan	CM Requirements Report	CM Design Report	CM Implementation Report	CM Integration Report	CM Validation Report	CM Installation Report	CM Change Report

Figure 3. Class 1E Software Life Cycle Activities

4.2. Verification Audit Process Description

The audit process proposed in the guidelines consists of four phases:

- Audit Preparation Phase,
- Audit Performance Phase,
- Audit Reporting Phase, and
- Audit Close-out Phase.

The majority of the effort is expended during the Audit Preparation Phase, which is performed before the on-site visit. This phase consists of the definition of the audit purpose, identification of the audit scope, and identification of the audit performance standards. During this phase, interactions with the vendor occur to obtain required audit material and to tailor the audit plan to the vendor's software development process. This phase is complete with the selection and orientation of the audit team. The next phase, the Audit Performance Phase, begins with the entry briefing or opening meeting, and includes performing the audit using the audit procedures and checklists prepared in the audit plan, the daily audit team meetings, and the daily briefings to vendor management. The Audit Reporting Phase begins with the audit exit briefing or closing meeting and concludes with the production of the Audit Report. It should be noted that the Audit Report is the only product produced by the audit team as part of the audit process. As such, this document is extremely important since it is the only evidence that the audit actually occurred. The Audit Close-out Phase consists of interaction between the NRC and the vendor as findings are reviewed and corrective actions are planned and analyzed.

4.3. Verification Audit Processes and Products Audited

As shown in Figure 3, the set of software development activities performed during the software development process, in accordance with a particular vendor software life cycle, uses several processes to produce software products. It should be noted that some of the processes are totally contained within a specific set of software development activities, such as the process that is used to produce the software V&V Plan. On the other hand, some processes span several sets of software development activities. A case in point is the software configuration management process, which spans all sets of software development activities. In all cases, the processes used for software development produce interim software products that can each be evaluated. The verification audit field test described in this paper looked at the following software processes and products: software planning, software requirements, software safety requirements, software safety analysis, software verification and validation, software configuration management, software design, software implementation, software test, and hardware and software integration activities.

4.4. Verification Audit Sample Checklists

Each of the processes and products shown in Figure 3 can and should be evaluated using a checklist or set of questions developed for the audit plan using the guidance provided by the V&V Guidelines. Portions of sample checklists used to evaluate software products are shown below. These include partial samples of the following software products; namely, a software development plan, a software requirements specification, and a software code safety analysis.

More detailed information can be found in the field test verification audit plan. (See Persons 1993a.)

4.4.1. Sample Software Development Plan Checklist

The following is a sample of a verification audit checklist that can be used to audit a software development plan for a Class 1E software project.

Software Development Plan Checklist

The Software Development Plan is the plan that guides the technical aspects of the development project. It will specify the life cycle that will be used, and the various technical activities that take place during that life cycle. All methods, tools and techniques which are required in order to perform the technical activities will be identified.

1. **Life Cycle Process Questions.**
   a. Is a software life cycle defined?
   b. Are the defined life cycle processes sufficient to provide confidence that a safe and adequate product will be produced?
   c. Are the inputs and outputs defined for each life cycle process?
   d. Is the source of each life cycle process input specified?
   e. Is the destination of each life cycle process output specified?
   f. Does each life cycle phase require a safety analysis?
   g. Does each life cycle phase include a requirement for an audit at the end of the phase?

…

4.4.2. Sample Software Requirements Checklist

The following is a sample of a verification audit checklist that can be used to audit a software requirements specification (SRS) for a Class 1E software project.

Software Requirements Specification (SRS) Checklist

The Software Requirements Specification (SRS) documents all the software requirements. These come from the specific system or product design and the specific system or product hazard analysis.

1. **User Characteristics Questions.**
   a. Is each category of user identified in the SRS?
   b. Is the expected experience level of each category of user defined?
   c. IS the training requirement for each category of user defined?

▪ ▪ ▪

6. **Performance Requirements Questions.**
   a. Are all static performance requirements fully described?
   b. Are all system timing requirements included in the SRS?
   c. Are the timing requirements specified numerically?
   d. Are timing requirements expressed for each mode of operation?

…

4.4.3. Sample Code Safety Analysis Checklist

The following is a sample of a verification audit checklist that can be used to perform a code safety analysis audit for a Class 1E software project.

Code Safety Analysis Checklist

The purpose of the safety analysis is to identify any errors or deficiencies in the code which could contribute to a hazard.

1. **Logic Questions.**
   a. Does the code logic correctly implement the safety-critical design criteria?
   b. Are design equations and algorithms correctly implemented in the code?
   c. Does the code correctly implement the error handling design?
   d. Does the code correctly implement the off-normal and emergency operations design?
   e. Is there convincing evidence that no code considered to be non-critical can adversely impact the function, timing, and reliability of the safety-critical code?
   f. Is there convincing evidence that any interrupts that may be included in the code will not take precedence over or prevent the execution of safety-critical code modules?
2. **Data Questions.**
   a. Are the definition and use of data items in the code consistent with the software design?
   b. Is each data item in the code explicitly typed?
   c. Is there a convincing argument that no safety-critical data item can have its value changed in an unanticipated manner, or by an unanticipated module?
   d. Is there a convincing argument that no interrupt can destroy safety-critical data items?

…

4.5. Verification Audit Report

The verification audit report is the only product of the audit team and serves as evidence that the audit was performed. It encapsulates the audit scope, purpose, audit process used, and associated information. Based on the V&V Guidelines, a sample audit report was generated and used for the field test verification audit. More specific information can be found in the verification audit report. (See Persons 1993b). A sample table of contents for a typical verification audit plan is shown below:

- Executive Summary
- Introduction
- Software Verification Audit Description
  - Scope
  - Purpose
- Definitions and Acronyms
- Identification of the Auditors
- People Contacted
- Software Verification Audit Summary
- Software Verification Audit Results
  - Findings
  - Observations
  - Concerns
- Recommendations
- Positive Indications
- Software Verification Audit Procedure
- References
- Attachments
  - Audit Plan
  - Audit Schedule
  - Entrance Briefing Viewgraphs
  - Vendor Approach to Software Development
  - Completed Checklists.

Most of the table of contents entries are self-explanatory; however, it should be noted that the Software Verification Summary includes all the processes and products that were analyzed during the audit. In addition, the Verification Audit Procedure section should describe the audit procedures and checklists that were used to perform the verification audit.

4.6. Verification Audit Lessons Learned

Audits are of necessity conducted during a very limited time frame. The V&V Guidelines provide guidance as to how to increase the effectiveness of audits for safety-critical software performed within the limited time available. The purpose of V&V Guidelines, audit plans, audit procedures, and audit checklists is to aid the audit team in the final evaluation of the risk associated with the safety-critical software in question.

Focus on the software development process and its related products provides visibility into the software planning, management, analysis, design, implementation, and testing phases and greatly increases the understanding of the various software development and product assurance processes. It provides a valuable framework for the auditors as they make their determination as to whether the licensee/vendor complies with applicable standards for Class 1E software for nuclear power plants. The following are lessons learned which, when incorporated into the V&V Guidelines, should yield a more powerful assessment tool for verification audits that will serve both Class 1E software developers and NRC regulators alike.

- A pre-audit visit should be performed to establish the scope and purpose of the audit.
- Review of the vendor's software development process should be completed prior to developing the verification audit plan.
- Review of available system and software development documentation should be completed in advance of the on-site visit.
- Tailoring of the audit process to the vendor's software development process should be completed as part of the audit plan development.
- The auditee should be given advance notification of the materials to be reviewed by the audit team.
- Several verification audits should occur to assess the software processes and products as they are created. Start the audit process early in software development life cycle with a review of the planning documentation.
- The audit team should include specialists in the specific processes and/or products under evaluation. The composition of the audit team can and should change from one verification audit to another.
- The audit process is labor-intensive and attention to detail is required.
- A two-level evaluation process is desirable. The first level screens the processes or products to determine if further evaluation is appropriate.
- The audits need to be consistent across time and companies. Knowledge of past audits of the same process or product is desirable and helpful in performing the current verification audit.
- Easy access to standards is helpful.
- Automated assistance for the audit team in the creation of daily activity summary and interim documentation would make the audit process more efficient.

5. FUTURE VISION OF CLASS 1E EVALUATION

The evaluation of Class 1E software by the regulator should be consistent with the developer's life cycle. Many different life cycle models exist and are used by software development companies, but they include the same basic activities. These models differ primarily in the ordering of activities through time. As a result, the regulator can concentrate on evaluating the activities as they occur during the developer's life cycle. Because of this variation between life cycle models, evaluation means must exist for each different activity in the life cycle. This argument is the basis for the vision presented in this paper.

This vision of future Class 1E software evaluation suggests a change in perspective that involves computer assistance for the evaluation process. Human Factors studies and task analysis experiences suggest that there is a general pattern for automating any process. As applied to the regulatory evaluation process, the following steps are useful:

- Control the complexity of the audit process.
- Define and separate the audit activities by software life cycle activity.
- Define activity-specific audit procedures.
- Develop support tools to encourage the use of consistent evaluation practices.
- Develop an integrated environment to assist in the audit process.

5.1. Automation Issues

To be successful in introducing automated support into auditing Class 1E software, an integrated approach is suggested that includes the auditor, methods, tools, and data. Goals for the support environment include increased accuracy; consistency and productivity; ability to customize to specific evaluation needs; and acceptance by the auditors.

Automated support for audits or evaluations raises several issues that needed to be addressed.

- Increasing consistency among auditors.
- Reducing the time and effort required to perform an audit.
- Training team members in the use of the evaluation process and supporting tools.
- Determining the processes and products to be audited.
- Defining the audit process.
- Tailoring the evaluations to life cycles used by different companies.
- Maintaining consistency among tools.
- Providing a common user interface.
- Maintaining a data base of past audits.
- Maintaining a data base of important guidance and standards.
- Providing security and access control.

5.2. Automation Architecture

A conceptual architecture for an automated approach is shown in Figure 4. Suppose that an audit is imminent. Through the graphical user interface the auditor selects the developer's activities to be evaluated. The automated support provides appropriate tools, standards, and guidance for use in the specific audit and then guides the auditor through that process.

This conceptual architecture has two objectives. The first objective is to provide step-by-step guidance to the evaluator in the use of the evaluation process. The second objective is to provide automated support for preparing the auditor's documentation, including the audit plan, supporting data, the audit report, and information that is to become part of the historical data base. A cost/benefit analysis of such an automated approach is in progress.

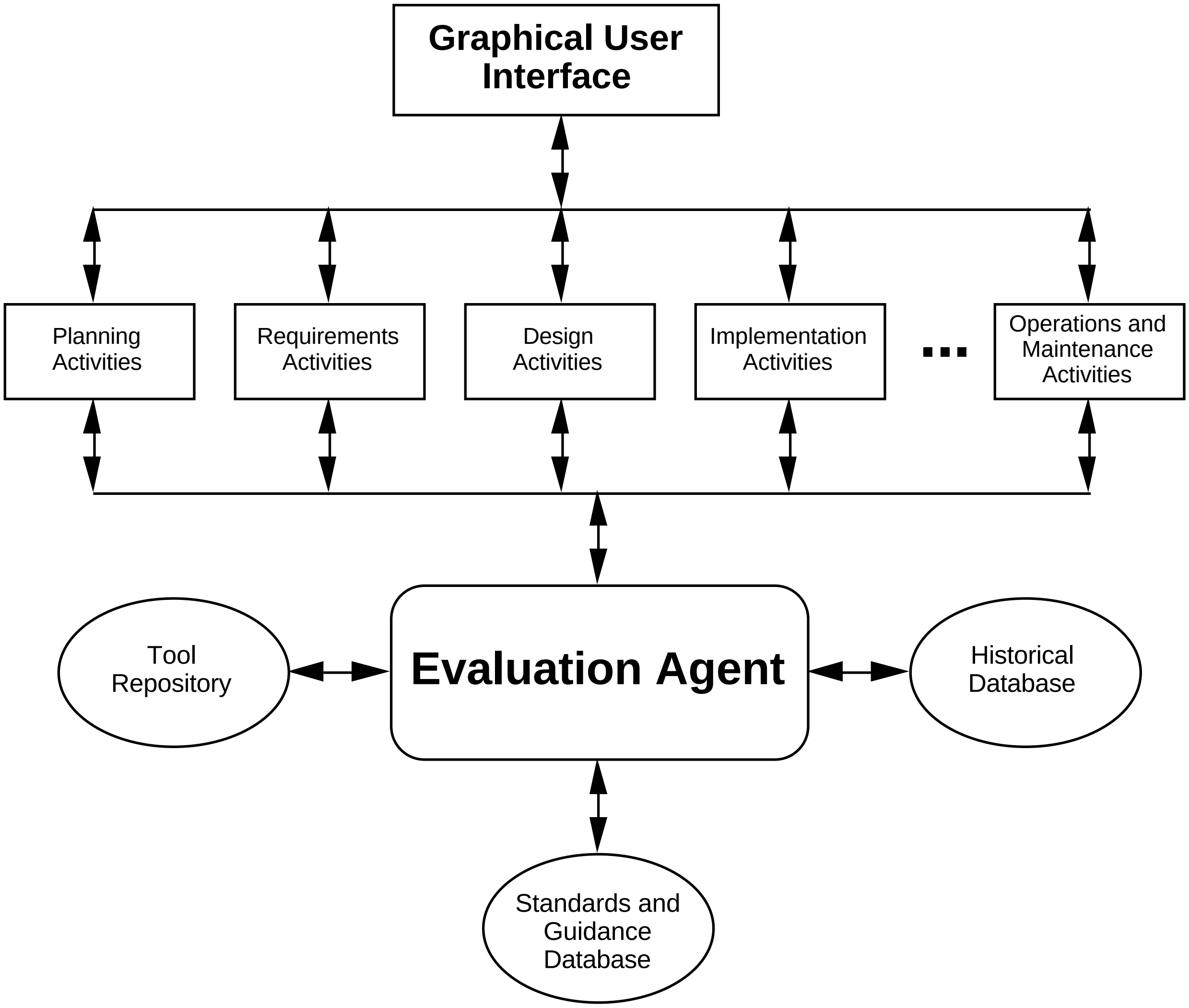


Figure 4. Conceptual Architecture of the Evaluation Assistant

GLOSSARY

audit. An independent evaluation of software products or processes to ascertain compliance to standards, guidelines, specifications, and procedures based on objective criteria that include documents that specify:

(1) the form or content of the products to be produced
(2) the process by which the products shall be produced
(3) how compliance to standards or guidelines shall be measured. (IEEE Std. 610.12).

Class 1E software. The safety classification of the electric equipment and systems that are essential to emergency reactor shutdown, containment isolation, reactor core cooling, and containment and reactor heat removal, or are otherwise essential in preventing significant release of radioactive material to the environment. Software used in Class 1E systems is referred to as Class 1E software. (IEEE Standard 379).

Class 1E system. The safety classification of the electrical equipment and systems that are essential to emergency reactor shutdown, containment isolation, reactor core cooling and containment and reactor heat removal, or are otherwise essential in preventing significant release of radioactive material to the environment. (IEEE Std. 379).

evaluation. Determination of fitness for use. (IEEE Std. 1074).

procedure. (1) A course of action to be taken to perform a given task. (2) A written description of a course of action as in (1); for example, a documented test procedure. (IEEE Std. 610.12-1990).

process. (1) A sequence of steps performed for a given purpose; for example, the software development process (IEEE Std. 610.12). (2) A function that must be performed in the software life cycle. A process is composed of activities. (IEEE Std. 1074).

product assurance. The software developer's activities which relate to verification and validation, testing, software quality assurance (SQA), software configuration management (SCM), and safety analysis.

review. An evaluation of software element(s) or project status to ascertain discrepancies from planned results and to recommend improvement. This evaluation follows a formal process (for example, management review process, technical review process, software inspection process, or walkthrough process). (IEEE Std. 1028).

safety. (1) Freedom from those conditions that can cause death, injury, occupational illness, or damage to or loss of equipment or property, or damage to the environment (MIL-STD 882C). (2) The expectation that a system does not, under defined conditions, lead to a state in which human life, limb and health, economics or environment are endangered. Note: For system safety, all causes of failures which lead to an unsafe state shall be included; hardware failures, software failures, failures due to electrical interference, due to human interaction and failures in the controlled object. Some of these types of failure, in particular random hardware failures, may be quantified using such measures as the failure rate in the dangerous mode of failure or the probability of the protection system failing to operate on demand. The system safety also depends on many factors which cannot be quantified but can only be considered qualitatively. (IEC 65A (Secretariat) 122).

safety-critical software. (1) Software whose inadvertent response to stimuli, failure to respond when required, response out-of-sequence, or response in unplanned combination with others can result in an accident. Also, software that is intended to mitigate, or recover from the result of an accident (IEEE P1228 Draft E). (2) Software which ensures that a system does not endanger human life, limb and health, or the economics of environment of the capital equipment and control. (IEC 65A (Secretariat) 122).

software development process. (1) The process by which user needs are translated into a software product. The process involves translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes, installing and checking out he software for operational use. Note: These activities may overlap or be performed iteratively (IEEE Std 610.12). (2) A set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases, user manuals, etc.). (CMU/SEI-91-TR-24).

software life cycle (SLC). A project-specific, sequenced mapping of activities. (IEEE Std. 1074).

software reliability. The probability that software will not cause the failure of a system for a specified time under specified conditions. The probability is a function of the inputs to and use of the system as well as a function of the existence of faults in the software. The inputs to the system determine whether existing faults, if any, are encountered. (IEEE Std. 982.1).

software product. (1) The complete set of computer programs, procedures, and possibly associated documentation and data designated for delivery to a user. (2) Any of the individual items in (1). (IEEE Std. 610.12).

task. The smallest unit of work subject to management accountability. A task is a well-defined work assignment for one or more project members. Related tasks are usually grouped to form activities. (IEEE Std. 1074).

validation. The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements. (IEEE Std. 610.12).

verification. The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. (IEEE Std. 610.12).

verification and validation. The process of determining whether the requirements for a system or component are complete and correct, the products of each development phase fulfill the requirements or conditions imposed by the previous phase, and the final system or component complies with specified requirements. (IEEE Std. 610.12).

BIBLIOGRAPHY

Andriole, S. J., Editor, *Software Validation, Verification, Testing and Documentation*, Petrocelli Books, 1986.

AFSC/AFSLC Pamphlet 800-5, Acquisition Management: Software Independent Verification and Validation (IV&V), 1988.

ANS-10.4-1987, *American National Standard for the Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry*.

American Society of Mechanical Engineers.

ASME NQA-1-1989, *Quality Assurance Program Requirements for Nuclear Facilities*.

ASME NQA-2-1989, Proposed Addition to ANSI/ASME-NQA-2, Part 2.7, *Quality Assurance Program Requirements for Nuclear Facility Applications*, Draft 3.3.

CMU/SEI-91-TR-24, *Capability Maturity Model for Software*, Version 1.1, February 1993.

DO-178B, *Requirements and Technical Concepts for Aviation*, Draft Revision to *Software Consideration in Airborne Systems and Equipment Certification*.

DOD-STD-2168, Military Standard, *Defense System Software Quality Program*, 29 April 1988.

Institute of Electrical and Electronic Engineers.

IEEE-ANS-7-4.3.2-1982, *Application Criteria for Programmable Digital Computer Systems in Safety Systems of Nuclear Power Generating Stations*.

IEEE 379-1977, *Application of the Single Failure Criterion to Nuclear Power Generating Station Class 1E Systems*.

IEEE 603-1991, *Criteria for Safety Systems for Nuclear Power Generating Stations*.

IEEE 610.12-1990, *IEEE Standard Glossary of Software Engineering Terminology*.

IEEE 730-1989, *IEEE Standard for Software Quality Assurance Plans*.

IEEE 828-1990, *IEEE Standard for Software Configuration Management Plans*.

IEEE 830-1984, *IEEE Guide to Software Requirements Specifications*.

IEEE 982.1-1988, *IEEE Standard Dictionary of Measures to Produce Reliable Software*.

IEEE 983-1986, *IEEE Guide for Software Quality Assurance Planning*.

IEEE 1012-1986, *IEEE Standard for Software Verification and Validation Plans*.

IEEE 1016-1987, *IEEE Recommended Practice for Software Design Descriptions*.

IEEE 1028-1988, *IEEE Standard for Software Reviews and Audits*.

IEEE 1042-1987, *IEEE Guide to Software Configuration Management*.

IEEE 1058-1987, *IEEE Standard for Software Project Management Plans*.

IEEE 1074-1991, *IEEE Standard for Developing Life Cycle Processes.*

IEEE 1228 Draft (1993), *IEEE Standard for Software Safety Plans*, Draft J.

IEEE 1298-1992, *IEEE Standard Software Quality Management System*.

International Electrotechnical Commission.

IEC Standard Publication 880-1986, *Software for Computers in the Safety Systems of Nuclear Power Stations*.

IEC 65A (Secretariat) 122 (1989), *Software for Computers in the Application of Industrial Safety-Related Systems*, August 1, 1991.

International Organization for Standardization.

ISO 9000-1987, *Quality Management and Quality Assurance Standards — Guidelines for Selection and Use*. This is also ANSI/ASQC Q90-1987, *Quality Management and Quality Assurance Standards — Guidelines for Selection and Use*.

ISO 9000-3, *Quality Management and Quality Assurance Standards — Part 3: Guideline for the Application of ISO 9001 to the Development, Supply and Maintenance of Software*.

ISO 9001, *Quality Systems — Models for Quality Assurance in Design/Development, Production, Installation, and Servicing*.

ISO 9002, *Quality Systems — Models for Quality Assurance in Production and Installation*.

ISO 9003, *Quality Systems Models for Quality Assurance in Final Inspection and Test*.

ISO 9004, *Quality Management and Quality Elements — Guidelines*.

MIL-HDBK-286, Military Handbook, *A Guide for DOD-STD-2168 Defense System Software Quality Program*, 14 December 1990.

MIL-STD 882C, *Military Standard System Safety Program Requirements*.

Nuclear Regulatory Commission.

NUREG-0493, *A Defense-in-Depth and Diversity Assessment of the RESAR-414 Integrated Protection System*.

NUREG-4640, *Handbook of Software Quality Assurance Techniques Applicable to the Nuclear Industry*, 1987.

Persons, Warren L., “Software Verification Audit Plan,” Lawrence Livermore National Laboratory, Livermore, CA, 1993a.

Persons, Warren L., “Software Verification Audit Report,” Lawrence Livermore National Laboratory, Livermore, CA, 1993b.